Application of quantum algorithms to the study of permutations and group automorphisms

Marianna Bonanome[1], Mark Hillery[1,2], and Vladimír Bužek[3]
[1]*Department of Mathematics, Graduate Center of the City University of New York, 365 Fifth Avenue, New York, NY 10016*
[2]*Department of Physics, Hunter College of the City University of New York, 695 Park Avenue, New York, NY 10021 and*
[3]*Research Center for Quantum Information,Slovak Academy of Sciences, Dúbravská cesta 9, 845 11 Bratislava, Slovakia*

We discuss three applications of efficient quantum algorithms to determining properties of permutations and group automorphisms. The first uses the Bernstein-Vazirani algorithm to determine an unknown homomorphism from Z_{p-1}^m to $Aut(Z_p)$ where p is prime. The remaining two make use of modifications of the Grover search algorithm. The first finds the fixed point of a permutation or an automorphism (assuming it has only one besides the identity). It can be generalized to find cycles of a specified size for permutations or orbits of a specified size for automorphisms. The second finds which of a set of permutations or automorphisms maps one particular element of a set or group onto another. This has relevance to the conjugacy problem for groups. We show how two of these algorithms can be implemented via programmable quantum processors. This approach open new perspectives in quantum information processing when both the data and the programs are represented by states of quantum registers. In particular, quantum programs that specify control over data can be treated using methods of quantum information theory.

PACS numbers: 03.67.Lx

I. INTRODUCTION

Group theory has proven to be a useful arena in which to explore the application of quantum algorithms, that is implementation of fundamental concepts of quantum theory for design of efficient algorithms. Perhaps the most prominent example of this is the algorithm for the hidden subgroup problem [1, 2, 3, 4, 5]. In this problem, we are given a black box that evaluates a function, f. The function maps a finite group G to some finite set X. The function f separates [6] cosets [7] of some subgroup K of G. That is, f is constant on the left cosets of some subgroup, K of G and distinct for different cosets. The object is to use the black box (the function f) to determine a generating set for the subgroup K. This problem can be solved efficiently for Abelian groups by using a quantum algorithm. Specifically, Shor's algorithm for factoring is one particular example of this quantum algorithm. A quest for efficient quantum algorithms for the hidden subgroup problem for arbitrary groups would result in efficient quantum algorithms for problems such as the graph isomorphism problem and the shortest vector problem in lattices. Recently, progress has been made for certain non-Abelian groups, one example being the dihedral group [8, 9, 10] .

A. Quantum algorithms

Here we wish to show that quantum algorithms can be applied to the study of permutations and group automorphisms. If S is a finite set, a permutation, σ is a bijection mapping S into itself. A group automorphism is a permutation on a group that satisfies certain conditions. In particular, an automorphism, $\alpha : G \to G$ of a group, G, is a bijection of the group back into itself so that the group operation is preserved, i.e. if $g_1, g_2 \in G$, then $\alpha(g_1 g_2) = \alpha(g_1)\alpha(g_2)$. The set of automorphisms of a group, G, is itself a group, denoted by $Aut(G)$, with the group operation being function composition. If the order of the group, G, is N, then the automorphism group of G is a subgroup of the symmetric group on N objects, that is the group formed by permutations of N objects.

(1) Our first algorithm solves the following problem by making use of a variant of the Bernstein-Vazirani algorithm [11, 12]. Consider the cyclic group Z_p, where p is prime [13]. The automorphism group of Z_p is isomorphic to Z_{p-1} [14]. We consider a function $f : Z_{p-1}^m \to Aut(Z_p)$. In particular, we have that

$$f : (n_1, n_2, \ldots n_m) \to \alpha_{j_m}^{n_m} \alpha_{j_{m-1}}^{n_{m-1}} \ldots \alpha_{j_1}^{n_1}, \qquad (1)$$

where $n_k \in Z_{p-1}$ and $\alpha_{j_k} \in Aut(Z_p)$. We have a circuit whose inputs are an element of $Z_{p-1}^m, \overline{n} = (n_1, n_2, \ldots n_m)$, and an element, $l \in Z_p$, and whose output is an element of Z_p given by $\alpha(l)$, where $\alpha = f(\overline{n})$. Our task is to determine the automorphisms $\{\alpha_{j_k} | k = 1, 2, \ldots m\}$ with as few uses of the circuit as possible.

The other two algorithms are both variants of the Grover search algorithm [15].

(2) The first finds fixed points of permutations or automorphisms. Suppose the permutation σ on S has only one fixed point, that is, there is only one element $s \in S$ such that $\sigma(s) = s$. The object is to find the fixed point evaluating the permutation as few times as possible. This can clearly be used to find fixed points of automorphsims as well. By applying the same procedure to σ^n we can also find elements of S satisfying $\sigma^n(s) = s$. Permutations can be written as the product of disjoint cycles. By searching for points satisfying $\sigma^n(s) = s$, we are effectively searching for cycles of length n. Therefore, this algorithm can be used to find cycles of a specified length in a permutation. For an automorphism, this corresponds

to finding an orbit of a particular length.

(3) The final algorithm searches among permutations to find one with a specific property. In particular, suppose we have a set of permutations σ_k where $k = 1, 2, \ldots M$. Let s_1 and s_2 be two specfied elements of S, and suppose that only one of the permutations satifies the condition $\sigma_k(s_1) = s_2$. We want to find which one making use of as few function evaluations as possible. The corresponding problem for automorphisms is related to the conjugacy problem. For a group G, g_1 and g_2 in G are said to be conjugate if there is an element g_0 of G such that $g_1 = g_0 g_2 g_0^{-1}$. The mapping $\alpha_{g_0}(g) = g_0 g g_0^{-1}$ is an automorphism (automorphisms of this type are known as inner automorphisms). The conjugacy problem is, given two elements of a group, to determine whether they are conjugate to each other. The algorithm we are proposing can be applied to this problem.

Two of the algorithms we suggest are implemented by programmable quantum processors. A programmable quantum processor is a device that can perform several different functions on a quantum input state, which we shall call the data. The operation that is performed on the data is determined by a program, which is itself a quantum state. The advantage of such a device is that it is not necessary to construct a new quantum circuit for each operation, but only to change the program.

B. Programmable quantum processors

Programmable quantum processors have been the object of a number of recent studies. Nielsen and Chuang showed that the number of unitary operations such a device can deterministically perform is limited by the dimension of of the program space [16]. This led to the consideration of probabilistic [16, 17, 18, 19, 20, 21] and approximate processors [17, 18, 19, 22]. A probabilistic processor succeeds only part of the time, but we know when it does and when it does not. An approximate processor performs a set of operations to some specified level of approximation. Probabilistic and approximate processors do not suffer from the same limitations as deterministic ones do, and can perform larger sets of operations for a given program space dimension.

Another class of programmable devices consists of programmable measurement devices. These perform a measurement on an input state, with the measurement being specified by a program state. The first programmable measurement device was proposed by Dušek and Bužek [23], and since then a number of different types of these devices have been studied [24, 25, 26, 27, 28, 29].

One question that can be raised is whether having quantum programs gives one any advantage over simply having classical ones. A classically programmable quantum device can simply be thought of as one with a dial, with different settings of the dial leading to different quantum operations being applied to the input quantum state. There are several reasons that quantum programs can be advantageous. First, the program itself may be the result of a previous quantum computation. This allows programmable quantum processors to be chained together, with the output of one serving as the program of the next. A second reason is that the program may be intrinsically quantum. As an example of this, consider the programmable measurement device discussed in [28]. There the program consists of two qubits, one in the state $|\psi_1\rangle$ and the other in the state $|\psi_2\rangle$, both of which are unknown. The data consists of a qubit that is guaranteed to be in either $|\psi_1\rangle$ or $|\psi_2\rangle$, and our task is to determine which. Here, the program, consisting of two qubits, is intrinsically quantum; we are comparing our unknown qubit to a two-qubit string in order to determine which one it matches. Finally, a third reason is that if the programs are quantum, we can apply quantum information processing methods to the programs as well as the data. This is what we shall study here with two examples.

II. MODIFIED BERNSTEIN-VAZIRANI ALGORITHM

Before proceeding to the modified version of the Bernstein-Vazirani algorithm that solves the problem discussed in the Introduction, let us review what the original algorithm does. One is given a black box that evaluates a Boolean function, $f(x)$, whose argument, $x = x_{n-1} \ldots x_1 x_0$, is an n-digit binary number, and $x_j = 0, 1$ for $0 \leq j \leq n-1$. The function is of the form

$$f(x) = \sum_{j=0}^{n-1} x_j y_j + b = x \cdot y + b, \tag{2}$$

where y is a fixed, and unknown, n digit binary number, $b = 0$ or 1, and all additions and multiplications are modulo 2. The objective is to find y. Classically this requires $n+1$ function evaluations. One first finds b by evaluating the function for $x = 0$, and then one finds y_j by choosing x to be the string with $x_j = 1$ and all of the other digits equal to 0. Quantum mechanically only one function evaluation is required.

In the version of the algorithm considered here, the function is a mapping from Z_{p-1}^m to $Aut(Z_p)$ rather than one from Z_2^n to Z_2. As was stated in the introduction, the automorphism group of Z_p, where p is prime, is isomorphic to Z_{p-1}. This can be seen as follows. Each of the automorphism is completely determined by its action on 1, so let us denote by α_k the automorphism that satisfies $\alpha_k(1) = k$. Note that $\alpha_k(n) = nk \bmod k$. There are clearly $p-1$ such automorphisms, and any of the α_k for which k does not divide $p-1$ is a generator of the group $Aut(Z_p)$.

We now have the function $f : Z_{p-1}^m \rightarrow Aut(Z_p)$ specified by Eq. (1). This function can be implemented by a quantum circuit (see Fig. 1). It consists of gates that are controlled-unitary gates. The control line has inputs

from Z_{p-1}. The target line has inputs taken from Z_p. Elements of both of these groups are encoded as vectors from orthonormal bases. The action of the controlled-unitary gate is given by ($n \in Z_{p-1}$ and $l \in Z_p$)

$$|n\rangle|l\rangle \rightarrow |n\rangle|\alpha_k^n(l)\rangle, \tag{3}$$

for some α_k. We have $p-1$ different kinds of controlled-unitary gates, one for each automorphism in $Aut(Z_p)$. The circuit is composed of m of these gates. Consequently, there are m control lines, one for each of the inputs from Z_{p-1}^m. All of the unitary operators act on a common target line. The first gate corresponds to α_{j_1}, the second to α_{j_2}, etc. The inputs to the circuit are $|\overline{n}\rangle = |n_1\rangle \ldots |n_m\rangle$ corresponding to an element of Z_{p-1}^m and $|l\rangle$ corresponding to an element of Z_p. We shall denote the span of the vectors $|\overline{n}\rangle$ by $\mathcal{H}_P$, the program space, and the span of the vectors $|l\rangle$ by $\mathcal{H}_D$, the data space. The action of the circuit is

$$|\overline{n}\rangle|l\rangle \rightarrow |\overline{n}\rangle|\alpha(l)\rangle, \tag{4}$$

where $\alpha = f(\overline{n})$.

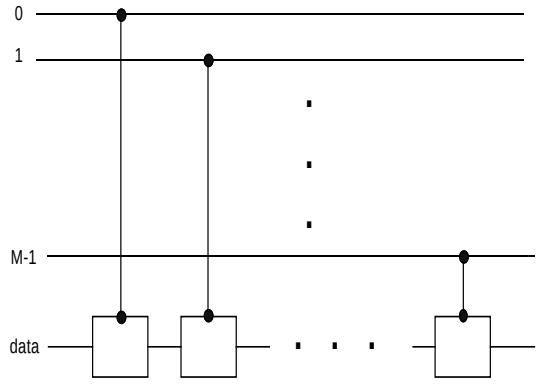

FIG. 1: The quantum circuit for the modified Berstein-Vazirani algorithm. Each control line is a qudit, where $d = p-1$, that is connected to a gate that implements a unitary operation on the data. The data space is spanned by the basis $\{|l\rangle | l = 1, 2, \ldots p-1\}$, where each basis state encodes a member of Z_p. If the k^{th} control line is in the state $|n\rangle$ the operation $U_{j_k}^n$ is performed, where U_{j_k} corresponds to the automorphism α_{j_k}. We have that $U_{j_k}|l\rangle = |\alpha_{j_k}(l)\rangle$.

For each of the automorphisms, α_k, define the vector in $\mathcal{H}_D$

$$|u_k\rangle = \frac{1}{(p-1)^{1/2}} \sum_{n'=0}^{p-2} e^{-2\pi i n'/(p-1)} |\alpha_k^{n'}(1)\rangle. \tag{5}$$

If U_k is the operator whose action is given by $U_k|l\rangle = |\alpha_k(l)\rangle$, then we have that

$$U_k|u_k\rangle = e^{2\pi i/(p-1)}|u_k\rangle. \tag{6}$$

We now choose one of the automorphisms, say α_{j_0}, that is a generator of $Aut(Z_p)$, i.e. any member of $Aut(Z_p)$ can be expressed as a power of α_{j_0}, and construct the vector $|u_{j_0}\rangle$. This vector is a simultaneous eigenvector of all of the operators U_k. In particular, if $\alpha_j = \alpha_{j_0}^s$, then

$$U_j|u_{j_0}\rangle = e^{2\pi i s/(p-1)}|u_{j_0}\rangle. \tag{7}$$

Therefore, there is a one-to-one correspondence between automorphisms and eigenvalues of $|u_{j_0}\rangle$.

We take the input state to the quantum circuit to be

$$|\Psi_{in}\rangle = \left(\frac{1}{(p-1)^{m/2}} \sum_{\overline{n} \in Z_{p-1}^m} |\overline{n}\rangle \right) \otimes |u_{j_0}\rangle. \tag{8}$$

The output state is then

$$|\Psi_{out}\rangle = \left(\frac{1}{(p-1)^{m/2}} \sum_{\overline{n} \in Z_{p-1}^m} e^{2\pi i \overline{y}\cdot\overline{n}/(p-1)} |\overline{n}\rangle \right) \otimes |u_{j_0}\rangle \tag{9}$$

where y_k is the unique integer between 0 and $p-2$ such that

$$\alpha_{j_k} = \alpha_{j_0}^{y_k} \tag{10}$$

We now note the following useful fact. Defining the vectors

$$|v_{\overline{r}}\rangle = \frac{1}{(p-1)^{m/2}} \sum_{\overline{n} \in Z_{p-1}^m} e^{2\pi i \overline{r}\cdot\overline{n}/(p-1)} |\overline{n}\rangle, \tag{11}$$

where $\overline{r} \in Z_{p-1}^m$ and $\overline{r}\cdot\overline{n} = r_1 n_1 + \ldots r_m n_m$, we have that

$$\langle v_{\overline{q}} | v_{\overline{r}} \rangle = \delta_{\overline{q},\overline{r}}. \tag{12}$$

Therefore, by measuring the output of the control lines in the basis $|v_{\overline{r}}\rangle$, we can determine $\overline{y}$. This member of Z_{p-1}^m completely specifies the function f, i.e. we know which automorphism is performed by each control line in the circuit.

One use of the quantum circuit has allowed us to determine the unknown function f. Classically, m uses would be required. One would always send 1 into the target input, and 1 into one of the control inputs and 0 into the others. This would allow one to determine the automorphism corresponding to the control input into which 1 had been sent. Repeating this operation for each control line (m times) would allow one to determine f.

Finally, let us note that this algorithm solves a particular kind of hidden subgroup problem, which is rather different from the usual one. The mapping f is a homomorphism from Z_{p-1}^m to $Aut(Z_p)$, and so its kernel is a subgroup of Z_{p-1}^m, and its image is a subgroup of $Aut(Z_p)$. One can view the algorithm we have just presented as solving the problem of finding either the kernel or the image of the unknown homomorphism, f. For example, suppose we wish to find the kernel. Once we have found $\overline{y}$ with one use of the quantum circuit, the kernel is given by the elements of Z_{p-1}^m satisfying $\overline{y}\cdot\overline{n} = 0 \bmod (p-1)$.

III. FINDING FIXED POINTS

Let us now consider the following problem. We are given a black box that implements an unknown permutation on a finite set, S, i.e. $\sigma : S \to S$. We would like to find the fixed points of the permutation, that is those elements $s \in S$ such that $\sigma(s) = s$. An algorithm that solves this problem could also be used to find cycles. Any permutation can be written as the product of disjoint cycles. If we know that a permutation, σ, has one cycle of n and has no cycles whose size divides n, then we can find the cycle by applying the algorithm to σ^n, because the elements of the cycle are fixed points of σ^n. Group automorphisms are special kinds of permutations, so they too can be decomposed into disjoint cylces. In that case the cycles are known as orbits. Clearly an algorithm that solves the problem of finding fixed points of permutations, would also be capable of finding orbits of automorphisms.

Consider now a set S and an permutation σ of that set that has only one fixed point, s_0. We shall show how we can use a modified Grover algorithm to find the fixed point. The elements of the set are encoded into an orthonormal basis of a Hilbert space, $\mathcal{H}_S$ whose dimension is equal to the number of elements in the set, N. We shall be operating in the space $\mathcal{H}_a \otimes \mathcal{H}_b$ where both $\mathcal{H}_a$ and $\mathcal{H}_b$ are copies of $\mathcal{H}_S$. Define the state

$$|v\rangle_{ab} = \frac{1}{\sqrt{N}} \sum_{s\in S} |s\rangle_a \otimes |s\rangle_b, \tag{13}$$

and the operator $U_\sigma|s\rangle = |\sigma(s)\rangle$. We begin the iteration that amplifies the amplitude of the state $|s_0\rangle$ by preparing the state

$$|\Psi_{in}\rangle_{ab} = (U_\sigma \otimes I_b)|v\rangle_{ab}. \tag{14}$$

We now need an operator that will perform the amplification of the desired state. This operator consists of two parts, one part flips the sign of the target state and this is followed by what Grover called the "inversion about the mean". The operator that performs the sign flip is $U_{id} = I_{ab} - 2P_{id}$, where

$$P_{id} = \sum_{s\in S} (|s\rangle_a\langle s|) \otimes (|s\rangle_b\langle s|). \tag{15}$$

Clearly, $U_{id}|s_1\rangle_a|s_2\rangle_b = |s_1\rangle_a|s_2\rangle_b$ if $s_1 \neq s_2$, and $U_{id}|s\rangle_a|s\rangle_b = -|s\rangle_a|s\rangle_b$. The second operator is given by

$$U_w = (U_\sigma \otimes I_b)(I_{ab} - 2|v\rangle_{ab}\langle v|)(U_\sigma^{-1} \otimes I_b). \tag{16}$$

Note that in order to implement this operator, we need black boxes that implement both U_σ and U_σ^{-1}.

We now apply $-U_wU_{id}$ to $|\Psi_{in}\rangle_{ab}$ $O(\sqrt{N})$ times. This has the effect of bringing the amplitude of the state $|s_0\rangle_a|s_0\rangle_b$ close to one and suppressing the amplitudes of all of the other states, so that if the state is measured in the basis $\{|s_1\rangle_a \otimes |s_2\rangle_b|s_1, s_2 \in S\}$ the probability to find $|s_0\rangle_a|s_0\rangle_b$ is close to one. We shall not prove this statement here, because the argument is standard (see Ref. [30]), and we shall present a more detailed discussion of the Grover algorithm in the next section. It is useful, however, to see the effect of one application of $-U_wU_{id}$. We have that

$$\begin{aligned} -U_wU_{id}|\Psi_{in}\rangle_{ab} &= \frac{1}{\sqrt{N}}\left[\left(3 - \frac{4}{N}\right)|s_0\rangle_a|s_0\rangle_b \right. \\ &+ \left. \left(1 - \frac{4}{N}\right)\sum_{s\neq s_0} |\sigma(s)\rangle_a|s\rangle_b\right]. \end{aligned} \tag{17}$$

Note that the effect has been to increase the amplitude of $|s_0\rangle_a|s_0\rangle_b$ and decrease the amplitude of all of the other states in the superposition.

If one does not know the number of fixed points, quantum counting can be used to find it [31, 32]. This procedure uses phase estimation to find the eigenvalues of the operator that implements the Grover iteration, and this allows one to determine the number of solutions, which in this case is the number of fixed points.

IV. DETERMINING WHETHER TWO ELEMENTS ARE AUTOMORPHIC IMAGES

Let $X = \{k|k = 0, \ldots N-1\}$ and suppose we have a set of permutations on X, $S = \{\sigma_j|j = 1, \ldots M\}$. In addition, if $x_0, y_0 \in X$ are specified elements of X, we shall assume that only one of the permutations σ_j satisfies the condition $\sigma_j(x_0) = y_0$. Our object is to find this permutation.

There are a number of problems for which the problem stated above forms a template. We are interested in the case in which the elements of the set X are elements of a group, and the permutations represent the actions of automorphisms, and S is a subgroup of the automorphism group of the original group. The object is then to find the automorphism for which one specified group element is the automorphic image of another specified element. The problem can be modified so that we ask whether or not there is an automorphism in the set S so that one specified group element is the automorphic image of another. The automorphisms may or may not be inner. An inner automorphism, α, is of the form $\alpha(g) = hgh^{-1}$, where g is an arbitrary element of the group G and h is fixed element of G. If they are, we are trying to determine whether group elements are conjugate to each other. If S is the subgroup of the automorphism group consisting of the inner automorphisms, then we are trying to determine whether two specified group elements are conjugate to each other, i.e. whether they are in the same conjugacy class. This is known as the conjugacy problem.

Another group theoretic problem which serves as a motivation to look at the above search problem for permutations is the Whitehead problem. In that problem, one is

trying to determine whether two elements of a free group are automorphic images of each other [33]. A generalized form of the Whitehead problem considers n-tuples. In particular, given two n-tuples consising of elements of the free group, $(g_1, g_2, \ldots g_n)$ and $(h_1, h_2, \ldots h_n)$, does there exist an automorphism of the group, α, such that $\alpha(g_j) = h_j$ for $j = 1, \ldots n$? A further variant is to restrict the set of automorphisms one is considering. In that case one would be interested in determining whether there is an automorphism in the allowed set that maps one specified n-tuple to a second specified one.

Classically, the only way to find the desired permutation is simply to try them one by one to see which one gives the result $\sigma_j(x_0) = y_0$. This would typically take of the order of M steps. What we want to show here is that by an application of a modified form of Grover's algorithm, we can, on a quantum computer, find the desired permutation in a number of steps that is of the order of $\sqrt{M}$.

We suppose that we have a quantum processor that causes the unitary operator V to be applied to the state vector $|\Psi\rangle$. The Hilbert space is a product of two others, $\mathcal{H} = \mathcal{H}_S \otimes \mathcal{H}_X$, where $\mathcal{H}_S$ is spanned by the orthonormal basis $\{|j\rangle_S | j = 1, \ldots M\}$, and $\mathcal{H}_X$ is spanned by the orthonormal basis $\{|x\rangle_X | x = 0, \ldots N-1\}$. The operator V has the following action

$$V|j\rangle_S|x\rangle_X = |j\rangle_S U_j|x\rangle_X, \tag{18}$$

where the operator U_j, acting on $\mathcal{H}_X$ implements the permutation σ_j, i.e. $U_j|x\rangle_X = |\sigma_j(x)\rangle_X$. What we see is that our processor is a controlled-U gate, with the unitary operators that it performs corresponding to the permutations in the set S. As we shall see shortly, besides a processor that implements V we shall also need one that implements V^{-1}.

We start the system in the state $|\Psi_{in}\rangle = |w\rangle_S|x_0\rangle_X$, where

$$|w\rangle = \frac{1}{\sqrt{M}} \sum_{j=1}^{M} |j\rangle_S. \tag{19}$$

We first apply V, and then apply the operator

$$\begin{aligned} Q \; = - \; & V(I - 2|w\rangle_S\langle w| \otimes |x_0\rangle_X\langle x_0|)V^{-1} \\ & (I - 2I_S \otimes |y_0\rangle_X\langle y_0|), \end{aligned} \tag{20}$$

a number of times n, where n is yet to be determined, to the initial state. At the end of the iteration procedure our state is $Q^n V|\Psi_{in}\rangle$, and we then measure the state in the basis $\{|j\rangle_S | j = 1, \ldots M\}$. With a probability close to one we will obtain the j corresponding to the desired permutation.

In order to see how this works, let us find the state after one iteration. We can assume, without loss of generatlity, that $j = 1$ corresponds to the desired permutation, We then have that

$$\begin{aligned} Q|\Psi_{in}\rangle \; = \; & V\frac{1}{\sqrt{M}}\left[\left(3 - \frac{4}{M}\right)|1\rangle_S \right. \\ & \left. + \sum_{j=2}^{M}\left(1 - \frac{4}{M}\right)|j\rangle_S\right] \otimes |x_0\rangle_X. \end{aligned} \tag{21}$$

We note that what has happened is that the probability of $j = 1$ has increased, and the probabilities of all of the other values of j has decreased. This is exactly what happens in the standard Grover algorithm. Further applications of Q will increase the probability of $j = 1$ further, as long as we do not do it too many times.

For a more detailed analysis, we note, again as with the standard Grover algorithm, that all of the action takes place in a two-dimensional subspace, and that Q is simply a rotation in that subspace. In particular, the subspace is the span of the two vectors $|v_1\rangle = |1\rangle_S|y_0\rangle_X$ and $|v_2\rangle = V|\Psi_{in}\rangle$. We find that

$$\begin{aligned} Q|v_1\rangle \; &= \; |v_1\rangle - \frac{2}{\sqrt{M}}|v_2\rangle \\ Q|v_2\rangle \; &= \; \left(1 - \frac{4}{M}\right)|v_2\rangle + \frac{2}{\sqrt{M}}|v_1\rangle. \end{aligned} \tag{22}$$

The vectors $|v_1\rangle$ and $|v_2\rangle$ are not orthogonal, so we define the vector

$$|v_1^\perp\rangle = \left(\frac{M}{M-1}\right)^{1/2}\left(|v_2\rangle - \frac{1}{\sqrt{M}}|v_1\rangle\right). \tag{23}$$

In terms of the basis $\{|v_1\rangle, |v_1^\perp\rangle\}$ the action of Q is given by

$$Q|v_1\rangle \; = \; \left(1 - \frac{2}{M}\right)|v_1\rangle - \frac{2}{\sqrt{M}}\left(1 - \frac{1}{M}\right)^{1/2}|v_1^\perp\rangle\,; \tag{24}$$

$$Q|v_1^\perp\rangle \; = \; \left(1 - \frac{2}{M}\right)|v_1^\perp\rangle + \frac{2}{\sqrt{M}}\left(1 - \frac{1}{M}\right)^{1/2}|v_1\rangle.$$

Defining the angle α as

$$\sin\alpha = \frac{2}{\sqrt{M}}\left(1 - \frac{1}{M}\right)^{1/2}, \tag{25}$$

we see that for any vector of the form

$$|\psi\rangle = \cos\theta|v_1\rangle + \sin\theta|v_1^\perp\rangle, \tag{26}$$

we have that

$$Q|\psi\rangle = \cos(\theta - \alpha)|v_1\rangle + \sin(\theta - \alpha)|v_1^\perp\rangle. \tag{27}$$

The procedure we have outlined starts in the state $|v_2\rangle$, which is close to, but not quite, orthogonal to $|v_1\rangle$. We want to rotate the state so that it becomes close to $|v_1\rangle$, because it is the $\mathcal{H}_S$ part of $|v_1\rangle$ that contains the information about which permutation has the desired property. This means that we want to iterate the process approximately $n = \pi/(2\alpha) \simeq \pi\sqrt{M}/4$ times. This is a considerable improvement over the roughly M steps we would have to perform classically.

We can also easily extend this algorithm to search for n-tuples that are automorphic images of each other as in the generalized Whithead problem. Let us consider the case $n = 2$. In particular, we now want to find the permutation, σ_j such that $\sigma_j(x_0) = y_0$ and $\sigma_j(x_1) = y_1$. Our Hilbert space is now $\mathcal{H} = \mathcal{H}_{S1} \otimes \mathcal{H}_{S2} \otimes \mathcal{H}_{X1} \otimes \mathcal{H}_{X2}$, where $\mathcal{H}_{S1}$ and $\mathcal{H}_{S2}$ are both copies of $\mathcal{H}_S$, and $\mathcal{H}_{X1}$ and $\mathcal{H}_{X2}$ are copies of $\mathcal{H}_X$. The operator $V \otimes V$ acts as

$$\begin{aligned}(V \otimes V)|j_1\rangle_{S1}|j_2\rangle_{S2}|x\rangle_{X1}|x'\rangle_{X2} \\ = |j_1\rangle_{S1}|j_2\rangle_{S2}U_{j_1}|x\rangle_{X1}U_{j_2}|x'\rangle_{X2}.\end{aligned} \tag{28}$$

Now define the state

$$|W\rangle_{S1S2} = \frac{1}{\sqrt{M}} \sum_{j=1}^{M} |j\rangle_{S1}|j\rangle_{S2}, \tag{29}$$

and the operators

$$\begin{aligned}P^{(x)}_{X1X2} &= |x_0\rangle_{X1}\langle x_0| \otimes |x_1\rangle_{X2}\langle x_1| \\ P^{(y)}_{X1X2} &= |y_0\rangle_{X1}\langle y_0| \otimes |y_1\rangle_{X2}\langle y_1|.\end{aligned} \tag{30}$$

For our initial state we now choose $|\Psi_{in}\rangle = |W\rangle_{S1S2}|x_0\rangle_{X1}|x_1\rangle_{X2}$. We begin by applying $V \otimes V$ to this state, and then applying the operator

$$\begin{aligned}Q \;=\; & -(V \otimes V)(I - 2|W\rangle_{S1S2}\langle W| \otimes P^{(x)}_{X1X2}) \\ & (V^{-1} \otimes V^{-1})(I - 2I_{S1S2} \otimes P^{(y)}_{X1X2}),\end{aligned} \tag{31}$$

approximately $\pi\sqrt{M}/4$ times. We then measure either the $\mathcal{H}_{S1}$ or $\mathcal{H}_{S2}$ part of the state in the computational basis to determine the permutation.

Note that in the algorithms being discussed in this section, the quantum search is being applied to the programs that implement the permutations. This would not be possible if those programs were classical.

V. CONCLUSION

We have presented three applications of efficient quantum algorithms to the study of group automorphisms. In addition to finding properties of automorphisms, two of them can be applied to find properties of permutations, i.e. finding fixed points and finding which permutation maps one particular set element onto another.

This last task is accomplished by doing a Grover search on the programs of a programmable quantum processor. This shows that there are advantages to using quantum programs that are themselves quantum states. In particular, one can apply quantum information processing to programs as well as data. We expect that further work in this direction could prove useful.

Acknowledgement
This work was supported in part by the European Union projects CONQUEST and QAP, by the Slovak Academy of Sciences via the project CE-PI/2/2005, by the project APVT-99-012304.

[1] R. Boneh and R. Lipton, in *Lecture Notes in Computer Science* (Springer Verlag, Berlin, 1995) p. 424.
[2] P. Høyer, Phys. Rev. A **59**, 3280 (1999).
[3] M. Mosca and A. Ekert, in *Lecture Notes in Computer Science 1509*, edited by Colin Williams, (Springer Verlag, Berlin, 1999) p. 174, and quant-ph/9903071.
[4] R. Jozsa, Computing in Science & Engineering **3**, 34 (2001), and quant-ph/0012084.
[5] Ch. Lomont, quant-ph/0411037.
[6] Given a group G, a subgroup $K \in G$, and a set X, we say a function $f : G \to X$ *separates* cosets of K if for all $g_1, g_2 \in G, f(g_1) = f(g_2)$ if and only if $g_1K = g_2K$.
[7] If G is a group, K a subgroup of G, and g an element of G, then $gK = \{gk : k \in K\}$ is a left coset of K in G, and $Kg = \{kg : k \in K\}$ is a right coset of K in G. Only when K is normal will the right and left cosets of K coincide.
[8] G. Kuperberg, SIAM J. Comp. **35**, 170 (2005), and quant-ph/0302112.
[9] O. Regev, quant-ph/0406151.
[10] D. Bacon, A. M. Childs, and W. van Dam, Chicago Journal of Theoretical Computer Science, no. 2, (2006), and quant-ph/0501044.
[11] E. Bernstein and U. Vazirani, Proc. 25th Annual ACM Symposium on the Theory of Computing (ACM Press, New York, 1993) pages 11-20.
[12] For a very nice discussion of the Berstein-Vazirani algorithm, and quantum algorithms in general, see R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca, Proc. R. Soc. Lond. A **454**, 339 (1998) and quant-ph/9708016.
[13] A group G is called *cyclic* if there exists an element g (the generator) in G such that, when written multiplicatively, every element of the group G is a power of g. Since any group generated by an element in a group is a subgroup of that group, showing that the only subgroup of a group G that contains g is G itself suffices to show that G is cyclic.
[14] Joseph J. Rotman, *An Introduction to the Theory of Groups* (Springer, New York, 1995).
[15] L. K. Grover, Phys. Rev. Lett. **79**, 325 (1997).
[16] M. A. Nielsen and I. L. Chuang, Phys. Rev. Lett. **79**, 321 (1997).
[17] A. Yu. Vlasov, e-print quant-ph/0103119.
[18] G. Vidal and J. I. Cirac, e-print quant-ph/0012067.
[19] G. Vidal, L. Masanes, and J.I. Cirac, Phys. Rev. Lett. **88**, 047905 (2002).
[20] M. Hillery, V. Bužek, and M. Ziman, Phys. Rev. A **65**, 022301 (2002).
[21] M. Hillery, M. Ziman, and V. Bužek, Phys. Rev. A **69**, 042311 (2004).
[22] M. Hillery, M. Ziman, and V. Bužek, Phys. Rev. A **73**, 022345 (2006).
[23] M. Dušek and V. Bužek, Phys. Rev. A **66**, 022112 (2002).
[24] J. Fiurašek, M. Dušek, and R. Filip, Phys. Rev. Lett. **89**, 190401 (2002); J. Fiurašek and M. Dušek, Phys. Rev. A

69, 032302 (20040.

[25] J. Soubusta, A. Černoch, J. Fiurašek, and M. Dušek, Phys. Rev. A **69**, 052231 (2004).

[26] M. Sasaki and A. Carlini, Phys. Rev. A **66**, 022303 (2002); M. Sasaki, A. Carlini, and R. Jozsa, Phys. Rev. A **64**, 022317 (2001).

[27] G. M. D'Ariano and P. Perinotti, Phys. Rev. Lett. **94**, 090401 (2005).

[28] J. Bergou and M. Hillery, Phys. Rev. Lett. **94**, 160501 (2005).

[29] A. Hayashi, M. Horibe, and T. Hashimoto, Phys. Rev. A **73**, 012328 (2006); Phys. Rev. A **72**, 052306 (2005).

[30] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, 2006) chapter 6.

[31] G. Brassard, P. Høyer, and A. Tapp, *Lecture Notes in Computer Science 1443* (Springer Verlag, Berlin, 1999) p. 820, and quant-ph/9805082.

[32] M. Mosca, in *Proceedings of International Workshop on Randomized Algorithms, Workshop of Mathematical FOundations of Computer Science* edited by R. Freivalds, (Brno, Czech Republic 1998), available at http://eccc.hpi-web.de/eccc-local/ECCC-LectureNotes/randalg/inde

[33] Roger C. Lyndon and Paul E. Schupp, *Combinatorial Group Theory* (Springer-Verlag, Berlin, 2001).